# 专题讨论：分治方法

AC_Aerolight

2013.4.28~2013.4.29 @ Fuzhou

# Preface

- 今天首先讨论的是一种特殊的分治方法，在OI界初见于陈丹琦2008年的集训队作业中，因此也被称为CDQ分治。
- 随后将讨论一些利用了分治思想的其他算法。
- 这节课的目的是科普，因此题目会较简单，讲解也会比较详细。如果对该算法或对特定问题已有深入了解可以跳过不听，但不要打扰其他同学。
- Let’s begin.

# Preface

- 先看几个常见的递归复杂度分析。
  - $T(n)=2T(n/2)+O(kn)$的解是$T(n)=O(kn \log n)$
    - Master Theorem
  - $T(n)=2T(n/2)+O(kn \log n)$的解是$T(n)=O(kn \log^2 n)$
  - $T(n)=2T(n/2)+O(k)$的解是$T(n)=O(kn)$
    - 一棵N个节点的线段树上有几个节点？
  - K是一个和N无关的多项式

# Intro：归并排序

- 给定排列P，求排列的逆序对数量。
- P的长度<=100000。
- 要求O(nlogn)

- 定义归并排序过程Merge(l,r)
- Merge(l,r)
  - Merge(l,mid)
  - Merge(mid+1,r)
  - Count(l,mid,mid+1,r)
- 只需要考虑左右两段之间造成的逆序对，段内的逆序对由递归解决

# [NOI2007] Cash

- 有两种金券，金券按比例交易：买入时，将投入的资金，购买比例为Rate[i]的两种金券；卖出时，卖出持有的一定比例的金券。已知未来n天两种的金券价格A[i]、B[i]，初始资金为s，求最大获利。
- 为使获利最大，交易时显然应该全部买进或卖出。
- 1<=n<=100000

# [NOI2007] Cash

- 简要做法：

- 令F[i]表示第i天获得的最大B卷数量。
- 枚举上一次交易日j
- F[i]=max{ans,Rate[j]*F[j]*A[i]+F[j]*B[i]}/(rate[i]*a[i]+b[i])
- O(n^2)

- 观察括号内的表达式，可以发现决策J优于决策K的条件：(rate[j]*f[j]-rate[k]*f[k])/(f[j]-f[k])>-b[i]/a[i]
- 上面这个式子的左边，一般记成slope(j,k)

# [NOI2007] Cash

- Slope(j,k) > -b[i]/a[i]
- 令G[i] = rate[i]*f[i]，在二维平面上定义点Xi=(Fi,Gi)
- Slope(j,k)就是通过Xj和Xk的斜率

- 维护一个点集X，支持以下两个操作：
  - 1)在第一象限的任意位置插入一个点
  - 2)给定负数斜率K，求所有斜率为K且过点集中任意点的直线在Y轴上的最大截距
- 操作2最终用到的点都会在点集的上凸壳上
  - 维护点集X的凸包，支持动态插入和斜率查询
  - 平衡树结构 O(nlogn)

# [NOI2007] Cash

- 算法存在的问题
  - 边界情况众多
  - 难写难调
- 观察
  - 操作2中，提供的斜率值是-bi/ai，可以预处理得到而和F的取值无关
  - 这意味着在插入节点Xi时，已经可以确认它对询问j(j>i)带来的影响
- 引入分治思想

# [NOI2007] Cash

- 定义过程Solve(L,R)
- 假设运行Solve(l,r)可以得到F[l]到F[r]的值。
  - [L,mid]区间里的询问，可以直接递归Solve(l,mid)解决。
  - [mid+1,r]区间里的询问K，会受到[mid+1,k]这些点的影响，以及[l,mid]的影响。前半部分可以递归解决。

- Solve(l,r)
  - 递归调用Solve(l,mid)
  - 整体考虑[l,mid]间的点对[mid+1,r]间询问的影响。
  - 递归调用Solve(mid+1,r)

# [NOI2007] Cash

- 整体考虑[l,mid]对[mid+1,r]的影响

- 给定点集X和一系列询问
  - 每个询问是一个负数斜率
  - 回答所有斜率符合且通过点集X中任意点的直线中，Y轴的最大截距是多少

- 只要考虑点集X的上凸包。
  - 对于每个询问，在凸包上二分即可。

- 这一步的复杂度是O(nlogn)，这里n=r-l。

# [NOI2007] Cash

- 时间复杂度？
- Solve(l,r)的复杂度是O(nlogn)
- T(n)=2T(n/2)+O(nlogn)
- T(n)=O(nlog^2n)

- 离最优化还有距离。

- 需要log n的地方
  - 1) 求点集凸包
  - 2) 二分答案

# [NOI2007] Cash

- 1) 点集凸壳
  - 合并两个凸壳的时间复杂度是O(n+m)
  - Solve(l,r)结束后返回[Xl..Xr]的凸壳
  - 单步O(n)，总体O(n log n)
- 2) 二分答案
  - 放弃二分，离线处理
  - 把点集凸壳和所有询问排序，用两个指针扫描
- 2’) 对询问排序
  - 提前对询问进行一次归并排序，存储所有中间结果
  - 为什么不能在Solve()时进行归并？
  - 预处理复杂度O(nlogn),主递归中单步O(n)
- T(n)=2T(n/2)+O(n),T(n)=O(nlogn)

# [NOI2007] Cash

- Solve(l,r)是递归函数
- 如何消除递归？
  - 手工栈模拟递归
  - 存储中间过程？
- T(n)的解不变，但常数减小。

- 总结
  - 1) 分治思想->只考虑跨立作用
  - 2) 段内影响忽略不计->问题离线化

# [WF2011] Machine Works

- 你的公司获得了一个厂房N天的使用权和一笔启动资金，你打算在这N天里租借机器进行生产来获得收益。
- 可以租借的机器有M台。每台机器有四个参数D,P,R,G。你可以在第D天花费P的费用（当然，前提是你有至少P元）租借这台机器，从第D+1天起，操作机器将为你产生每天G的收益。在你不再需要机器时，可以将机器卖掉，一次性获得R的收益。
- **厂房里只能停留一台机器。**
- 不能在购买和卖出机器的那天操作机器，但是可以在同一天卖掉一台机器再买入一台。
- 在第N+1天，你必须卖掉手上的机器。
- 求第N+1天后能获得的最大资金。

# [WF2011] Machine Works

- 数据范围
- 租借天数D<=10^9
- 初始资金C<=10^9
- 机器数N<=10^5
- 对于每个机器：
  - 租借日Di<=D
  - 买入价Pi和卖出价Ri满足1<=Ri<Pi<=10^9
  - 每日收益Gi满足1<=Gi<=10^9

# [WF2011] Machine Works

- 观察
  - 场地能放机器就一定要放么？

- 先将机器按照出售时间Di排序。
- 将所有时间离散化。
- 用Fi表示在时刻Di卖掉手上机器后最多剩下多少钱。

- Fi = Max (F[i-1], F[j]-P[j]+R[j]+G[j]*(D[i]-D[j]-1))
- 条件是F[j]>P[j]。
- O(n^2)

# [WF2011] Machine Works

- Fi = Max (F[i-1], F[j]-P[j]+R[j]+G[j]*(D[i]-D[j]-1))
  - F[j]>P[j]
- 令A[j]=F[j]-P[j]+R[j]-G[j]*D[j]-G[j]
- 那么F[i] = Max(F[i-1],A[j]+G[j]*D[i])

- 和上一题一样了。
- 注意到斜率D[i]是不变的，因此可以对整个[F1,Fn]进行分治。

- 复杂度O(nlogn)，和平衡树维护凸壳同阶

# [WF2011] Machine Works

- F[i] = Max(F[i-1],A[j]+G[j]*D[i])

- 在平面上有若干直线y=G[j]*x+A[j]
- 维护一个直线集，支持以下两类操作
  - 插入一次函数y=kx+b
  - 给定询问D，求所有函数在x=d上的最大值

- 维护一系列半平面交

- 对偶问题？

# [BOI2007] Mokia

- 有一个2000000*2000000的棋盘，每个格子上有一个数字A[x,y]，现在要执行两类操作：

- Add x y a：A[x,y] += a
- Query x0 y0 x1 y1：询问矩阵(x0,y0)-(x1,y1)内所有格子的数字和。

- Add操作数<=160000 Query操作数<=10000

# [BOI2007] Mokia

- 棋盘大小和询问数相差巨大，所以肯定要离散化。
- 二维线段树维护？
  - MLE+TLE

- 二维树状数组+Hash？
  - Hash常数过大
  - 空间怎么开？？

# [BOI2007] Mokia

- 和之前的想法类似，定义操作Solve(L,R)
- Solve(L,R)应当能够处理[L,R]之间的所有操作

- Solve(L,R)
  - Solve(l,mid)
  - 处理[l,mid]中操作对[mid+1,r]中操作的影响
  - Solve(mid+1,r)

- 如何处理？

# [BOI2007] Mokia

- 前半部分的Add对后半部分的Query造成的影响
- 给定带权点集P=(Xi,Yi)，Q个询问(x0,y0,x1,y1)，对于每个询问，输出在对应矩形内的点权之和。
  - 在列上对询问进行差分，将一个询问拆成2个
  - 所有点和询问按Y排序，线段树维护

  - 在两维上对询问进行差分，将一个询问拆成4个
  - 所有点和询问按Y排序，树状数组维护
- T(n)=2T(n/2)+O(nlogn)
- T(n)=O(nlog^2n)

# [Violet 3]天使玩偶

- 维护二维点集P，支持以下两个操作

- (1)插入点(x,y)
- (2)给定询问(x,y)，求点集中离询问点最近的点
- 距离定义为曼哈顿距离
- Dis(P1,P2)=|x1-x2|+|y1-y2|

- N,m<=300000
- X,y<=1000000

- k-d树能不能做

# [Violet 3]天使玩偶

- 去除Dis(P1,P2)的绝对值符号
  - 分4种情况讨论：左上，左下，右上，右下
  - 只需要考虑一种情况（答案在询问的左下）
- 维护点集P，支持以下操作
- (1)将(x,y)插入点集
- (2)给定询问(x,y)，求满足dis(p1,p2)=(x-x’)+(y-y’)=x+y-(x’+y’)最小的点
  - 问题转为求x’+y’最大的点
- 没有操作1的情况
  - 对x排序，对y维护树状数组

# [Violet 3]天使玩偶

- 定义操作Solve(l,r)，处理[l,r]之间的询问
- Solve(l,r)
  - Solve(l,mid)
  - 考虑[l,mid]中的点对[mid+1,r]中询问的影响
  - Solve(mid+1,r)

- 给定带权点集P(x,y)，权值为x+y，给出Q个询问(x,y)，查找x'<=x,y'<=y的最小x'+y'。
  - 退化为没有操作1的情况
  - 按X排序，对Y维护树状数组

- $T(n)=2T(n/2)+O(n\log n), T(n)=O(n\log^2 n)$

# 2D Partial Order

- 有N个人，每个人有两种能力值Pi,Qi。
- 如果Pi>Pj && Qi>Qj，称I比J有能力
- 现在要求出最长的一个序列A=(A1,A2,…,At)，满足Ai比Ai+1有能力

- N<=100000。
- 为了简单起见，P和Q都是1到N的排列。

- 把人按照Pi排序，问题变为求序列Qi的LIS
- $F[i] = Max(\{F[j] \mid j<i \ \&\& \ Q[j]<Q[i]\})+1$
- 用数据结构维护，做到O(nlogn).

# 3D Partial Order

- 有N个人，每个人有三种能力值Pi,Qi,Ri。
- 如果Pi>Pj && Qi>Qj && Ri>Rj，称I比J有能力
- 现在要求出最长的一个序列A=(A1,A2,...,At)，满足Ai比Ai+1有能力

- N<=40000。
- 为了简单起见，P,Q,R都是1到n的排列

# 3D Partial Order

- 首先可以按照Pi把人排个序。
- 现在要求的是满足i<j,Qi<Qj,Ri<Rj的最长序列。

- 用F[i]表示以第i个人结尾的最长序列
- F[i] = Max{F[j] | j<i,Qj<Qi,Rj<Ri } +1
- 怎么搞？

- 线段树套平衡树/可持久化线段树
- O(nlog^2n)

# 3D Partial Order

- 尝试在这个问题上进行分治。
- 定义过程Solve(l,r)，能够得到F[l]..F[r]的值
- Solve(l,r)
  - Solve(l,mid)
  - 处理[l,mid]中元素对[mid+1,r]中F[x]取值的影响
  - Solve(mid+1,r)

- $F[x] = Max\{F[j] \mid j<x, Qj<Qx, Rj<Rx\} +1$
  - 1) x在[l,mid]中：集体处理
  - 2) x在[mid+1,r]中：由递归解决

# 3D Partial Order

- 处理[l,mid]中元素对[mid+1,r]中F[x]取值的影响

- 维护带权点集X=(Qi,Ri) (l<=i<=mid)，权值F[i]
- 支持询问：给定点(Qj,Rj) (mid+1<=j<=r)
- 在点集X中寻找一个点(Qi,Ri)使得Qi<Qj且Ri<Rj，满足以上条件的点中取权值最大的。

- 离线处理
  - 将所有点和询问按Qi排序，按Qi顺序处理
  - 维护能够在一个位置填入数字和查询区间最大值的数据结构
  - 线段树或者平衡树

# 3D Partial Order

- 解法总结
  - 第一维：排序
  - 第二维：分治
  - 第三维：离线，数据结构
- 时间复杂度分析
  - T(n)=2T(n/2)+O(nlogn)
  - T(n)=O(nlog^2n)
- 再加一维？
- 先考虑一个简单情况。

# Simplified 4D Partial Order

- 有N个人，每个人有四种能力值Pi,Qi,Ri,Si。
- 如果Pi>Pj && Qi>Qj && Ri>Rj && Si>Sj，称I比J有能力
- 对每一个人，输出任意一个比他有能力的人编号，或声明没有人比他有能力。

- N<=40000
- 简单起见，P,Q,R,S都是1-n的排列。
- 树套树套树？
  - 怎么写……
- 树套树？

# Simplified 4D Partial Order

- 首先把元素按Pi排序。
- 对每个i，要判断是否有一个j满足 j<i,Qj<Qi,Rj<Ri,Sj<Si.
- 条件太多了，不好做。
- 问题等价于Min(Sj | j<i,Qj<Qi,Rj<Ri) < Si
- 求Min(Sj | j<i,Qj<Qi,Rj<Ri)
- 回忆上一题
  - F[i] = Max{F[j] | j<i,Qj<Qi,Rj<Ri }
- 用相同方法处理即可。

# 4D Partial Order

- 有N个人，每个人有三种能力值Pi,Qi,Ri,Si。
- 如果Pi>Pj && Qi>Qj && Ri>Rj && Si>Sj，称I比J有能力
- 现在要求出最长的一个序列A=(A1,A2,...,At)，满足Ai比Ai+1有能力
- N<=20000
- 树套树套树？
- 树套树？

# 4D Partial Order

- 将元素按照$P_i$排序。

- 用$F[i]$表示以第i个人结尾的最长序列
- $F[i] = \mathrm{Max}\{F[j] \mid j<i, Q_j<Q_i, R_j<R_i, S_j<S_i\} + 1$
- 没有办法转化了。
- 直接做：
  - 树套树套树->TLE+MLE
  - Hash动态节点，三维树状数组……->？

- 我们尝试一下分治。

# 4D Partial Order

- 定义Solve(l,r)，解决F[l]到F[r]的问题。
  - Solve(l,mid)
  - 考虑[l,mid]中的点对[mid+1,r]中F[]取值的影响
  - Solve(mid+1,r)

- 整体考虑
  - 给定带权点集X=(Qi,Ri,Si) [l<=i<=mid]，权值F[i]
  - 给定(r-mid)个询问(Qj,Rj,Sj) [mid+1<=j<=r]
  - 对于每个询问，回答点集中满足Qi<Qj,Ri<Rj,Si<Sj的最大权值。
- 离线操作
  - 按Qi排序。询问满足Ri<Rj,Si<Sj的点的最大权值

# 4D Partial Order

- 询问满足Ri<Rj,Si<Sj的最大权值
  - 线段树套平衡树?
  - 树状数组套平衡树?

- 再分治一次
- 定义分治过程Solve2(l,r),用于处理Solve(l,r)中[l,mid]中的点对[mid+1,r]中F[]值的影响。
- Solve2(l,r)
  - Solve2(l,mid)
  - 处理(l,mid)中点对(mid+1,r)中询问的影响
  - Solve2(mid+1,r)
- 和Solve(l,r)有什么区别?

# 4D Partial Order

- Solve(l,r)
  - Solve(l,mid)
  - **创建一个[l,r]的副本并按照Q[i]排序**
  - Solve2(l,r)
  - Solve(mid+1,r)

- Solve2()的工作
  - 维护带权点集X=(Ri,Si)，支持两个操作
  - 1) 将一个点(Ri,Si)插入X，权值F[i] (l<=i<=mid)
  - 2) 给出询问(Rj,Sj)，求一个权值最大且满足Ri<Rj,Si<Sj的点
- 在Solve2()上定义分治过程。

# 4D Partial Order

- Solve2(l,r)
  - Solve2(l,mid)
  - 处理[l,mid]的点对[mid+1,r]的询问的影响
  - Solve2(mid+1,r)

- 维护带权点集X=(Ri,Si) (l<=i<=mid)，权值F[i]
  - 支持询问：给定点(Rj,Sj) (mid+1<=j<=r)
  - 在点集X中寻找一个点(Ri,Si)使得Ri<Rj且Si<Sj，满足以上条件的点中取权值最大的。
- 很眼熟？
- 离线处理。
  - 同三维情况，排序后树状数组或线段树维护。

# 4D Partial Order

- 时间复杂度分析

- 定义T2(n)为Solve2(l,r)的时间复杂度。
- $T2(n)=2T2(n/2)+O(nlogn)$
- $T2(n)=O(nlog^2n)$
- 定义T(n)为Solve(l,r)的时间复杂度。
- $T(n)=2T(n/2)+T2(n)$
- $=2T(n/2)+O(nlog^2n)$
- $T(n)=O(nlog^3n)$

# 100D Partial Order

- 有N个人，每个人有100种能力值Pi1,Pi2,..,Pi100。
- 如果对于1<=k<=100有Pik>Pjk，称I比J有能力
- 现在要求出最长的一个序列A=(A1,A2,…,At)，满足Ai比Ai+1有能力

- N<=500。
- 简单起见，Px1,Px2..Px100都是1到n的排列。

- 99个log？
- 还想分治么？

# 100D Partial Order

- 枚举每对(i,j)，判断i是不是比j有能力

- 在构造出的图上求最长链

- O(100*n^2)

# [POI2011] Meteor

- 有N个国家开发小行星的轨道，设立了M个观察站。
- 观察站从1到M编号，每个观察站恰好属于一个国家。
- 第I和第I+1个观察站相邻，第M个和第1个相邻。
- 科学家预测在接下来的时间里会依次发生K场流星雨。
- 每场流星雨有一个区间[Li,Ri]，表示流星雨波及的区间是从第Li个观察站顺时针数到第Ri个。也就是说如果Li>Ri，指的是Li,Li+1,...,m,1,2,...,Ri-1,Ri。
- 每场流星雨有一个数量Ai，表示这场流星雨将会为波及的每个观察站提供Ai单位的陨石。
- 每个国家都拥有陨石收集数的目标Wi。对于每个国家，输出它在第几场流星雨后可以达到目标。
- N,M,K<=3*10^5,Ai,Wi<=10^9

# [POI2011] Meteor

- 样例输入

  3 5 // 国家个数N，空间站个数M
  1 3 2 1 3 // 每个空间站的归属Oi
  10 5 7 // 每个国家的目标Wi
  3 // 流星雨数量K
  4 2 4 // Li,Ri表示影响区间，Ai表示提供数量
  1 3 1
  3 5 2

样例输出

  3
  NIE // NIE表示达不到任务
  1

# [POI2011] Meteor

- 维护观察站信息
  - 线段树+Lazy-Tag
  - 差分化+树状数组
- 只有1个国家的情况
  - 每次流星雨后判断可行性
  - O(t*k*log m)
  - T是国家拥有的空间站数目
- 二分答案
  - 需要模拟O(N)次流星雨后判断可行性
  - O(t*k*log k*log m)
  - 为什么？

# [POI2011] Meteor

- 引入分治思想
- Solve(l,r)用于解决所有**答案落在[L,R]中的询问**
  - 两个参数够不够？
- Solve(l,r,S)
  - S是一个询问集合，表示需要处理的询问集合

- Solve(l,r,S)
  - 分割S，S1的答案在[l,mid]，S2的答案在[mid+1,r]
  - Solve(l,mid,S1)
  - Solve(mid+1,r,S2)

# [POI2011] Meteor

- 分割集合S
- 1) 用线段树模拟时刻Mid的情况
  - Mid是O(k)级别的
  - O(klogm)
- 2) 对于S中每个国家，查询已收集到的陨石数目
  - O(Σt*logm)
  - 由于Σt(全集)=n，一层递归的总复杂度O(nlogm)
  - 这一步的总复杂度是O(nlogklogm)
- (1)的时间复杂度？
  - T(x)=2T(x/2)+O(klogm)
  - T(x)=O(klogxlogm) -> AC?
  - **T(x)=O(x*klogm)！**

# [POI2011] Meteor

- TLE的原因在于(1)的单步复杂度和递归长度x无关。

- 维护全局线段树
- O(logm)模拟和**撤销**一次流星雨
  - 在运行Solve(l,r)之前，线段树存储了第1次到第l-1次流星雨的情况
  - 在运行Solve(l,r)之后，线段树存储了第1次到第r次流星雨的情况
- 用+[l,r]表示模拟[l,r]这段流星雨，-[l,r]表示撤销

- 如何写Solve(l,r,S)?

# [POI2011] Meteor

- Solve(l,r,S)
  - + [l,mid]
  - 分割点集S
  - - [l,mid]
  - Solve(l,mid,S1)
  - Solve(mid+1,r,S2)

- 模拟流星雨的时间复杂度
  - T(x)=2T(x/2)+**O(xlogm)**
  - **T(x)=O(xlogxlogm)**
- 整个算法的复杂度是O(klogklogm+nlogklogm)。
- AC。

# [POI2011] Meteor

- 并行分治
- 对于每个国家的询问，一开始的答案区间都是[1,n]。
- 同时对所有国家进行二分查询
- 主算法Solve执行一次整体二分，调用logk次
  - 每个国家答案区间的中点是midX
  - 询问第X个国家在midX时是否收集到Wx的陨石
- 对midX排序，离线处理
  - 顺序模拟所有流星雨，到第midX个时统计答案
  - 如果答案<Wx那么区间变成[mid+1,r]否则[l,mid]
- Logk次算法后，每个答案区间长度为1，直接输出
- T(n)=logk*(k*logm+n*logn)

# 矩阵乘法（梁盾）

- 给定n*n的矩阵A
- 给定Q个询问(x1,y1,x2,y2,k)，询问子矩形[x1,y1]~[x2,y2]中的K小数。

- N<=500
- Q<=60000
- **允许离线。**

# 矩阵乘法（梁盾）

- 数据结构
  - 线段树套线段树套树状数组……
- 单一询问的情况
  - 二分答案，判定可行性
  - 给定矩阵D[i,j]=(A[i,j]>=mid)，判定矩形内权值是否>k

- 准备分治
  - Solve(l,r,S)处理所有答案落在[l,r]之间的询问
  - 分割S到Solve(l,mid,S1)和Solve(mid+1,r,S2)中
- 构造D[i,j](mid)
  - 维护全局二维树状数组
  - 需要加的点只有(i,j) | L<=A[i,j]<=mid

# 矩阵乘法（梁盾）

- 时间复杂度统计
  - 单次操作可以视为log^2n
  - F(n)=2F(n/2)+O(log^2n)
  - F(n)=O(nlog^3n)
- 优化？

- 规避二维数据结构
  - 将询问和插入点集按x进行归并排序，维护y的树状数组
  - F(n)=2F(n/2)+O(nlogn)
  - F(n)=O(nlog^2n)

# [ZJOI2013] K大数查询

- 有n 个位置和m 个操作。操作有两种，每次操作如果是1 a b c 的形式，表示往第a 个位置到第b 个位置每个位置加入一个数c。如果操作形如2 a b c 的形式，表示询问从第a 个位置到第b 个位置，第c 大的数是多少。

- n,m<=50000
- 操作1中，c<=n
- 操作2中，c<=maxlongint

# [ZJOI2013] K大数查询

- 2 5
- 1 1 2 1
- 1 1 2 2
- 2 1 1 2
- 2 1 1 1
- 2 1 2 3
- 【样例输出】
- 1
- 2
- 1
- 第一个操作后位置1 的数只有1，位置2 的数也只有1。第二个操作后位置1的数有1、2，位置2 的数也有1、2。第三次询问位置1 到位置1 第2 大的数是1。第四次询问位置1 到位置1 第1 大的数是2。第五次询问位置1 到位置2 第3大的数是1。

# [ZJOI2013] K大数查询

- 常规的数据结构
  - 线段树套平衡树
  - 二分答案?
  - 外层线段树表示插入的数大小，内层表示坐标
  - 动态分配

- 单一询问的情况
  - 二分答案+维护线段树等等
- 准备分治
  - Solve(l,r,S)用于处理答案落在[l,r]之间的所有询问
  - 分割S集合

# [ZJOI2013] K大数查询

- 分割集合
  - 已知所有询问在L处的回答，求在MID处的回答
  - 所有的操作1中，只有l<=c<=mid的有作用
  - 按时间排序后维护树状数组
- 时间复杂度
  - 单步的复杂度均摊后不超过O(nlogn)级别
  - T(n)=2T(n/2)+O(nlogn)
  - T(n)=O(nlog^2n)

- 总结一下上面三题的共性

# Package

- 有N个物品，每个物品的重量是Wi,价值是Gi
- 每个物品只能取一个
- 给定Q个询问，每个询问由两个数(X,I)组成
- 给定最大容量为X的背包，使用除了第i件物品以外的所有物品，能够得到的最大价值之和

- N<=100
- 询问中的X<=10000
- Q<=1000000

- 怎么做？

# Package

- 离线操作
- 只有1个询问/所有询问的l相同的情况
  - 0/1背包
  - O(nm)

- 维护0/1背包
  - O(m)将一个物品放入背包。
  - 同样，定义+[l,r]表示把l到r的物品放入背包
  - +[l,r]的复杂度是O(nm)
- 准备分治
  - 对重量分治
  - **对物品ID分治**

# Package

- 定义Solve(l,r)，用于处理所有**l<=i<=r**的询问

- 定义Solve(l,r,S),用于处理所有l<=i<=r的询问
  - S是一个0/1背包，放进了**除了[l,r]之外**的其他物品
  - Solve(L,L,S)时，回答所有I=L的询问

- Solve(l,r,S)
  - Solve(l,mid,S+[mid+1,r])
  - Solve(mid+1,r,S+[l,mid])
- 时间复杂度分析
  - T(n)=2T(n/2)+O(nm)
  - T(n)=O(nmlogn)

# [HNOI2010] City

- 有一个N个点M条边的无向图，每条边有边权Wi
- 给定Q个操作(Ei,Zi)，表示将第Ei条边的费用改为Zi
- 每次操作之后，输出当前无向图的最小生成树权值

- N<=20000,M<=50000,Q<=50000,Wi,Zi<=10^8

- NO SPOILERS PLZ.

- 特殊情况：修改后边权不加。

# [HNOI2010] City

- 修改后边权不加的情况

- 修改(x,y)的权值时，最小生成树的**边构成**
  - 1）不变
  - 2）(x,y)加入最小生成树，另外一条边退出
- 2)发生的条件
  - 原MST上x到y路径上的最大值大于(x,y)的权值

- 维护MST
  - 支持加入一条边，删除一条边，查询路径最大值
  - Link-Cut Tree维护无根树

# [HNOI2010] City

- 修改后边权不减的情况
- ……?
- 预处理之后倒着做就行了。
- 不管怎么样，这个问题是**离线的**

# [HNOI2010] City

- 能不能对操作序列进行分治？
- 定义Solve(l,r)，用于处理L到R的操作并得到答案
- Solve(l,r)
  - Solve(l,mid)
  - 处理[l,mid]的加边对[mid+1,r]的询问的影响
  - Solve(mid+1,r)
- 一条边对一个询问的影响难以表示。

# [HNOI2010] City

- Solve(l,r)
  - Prepare(l,r)
  - Solve(l,mid)
  - Solve(mid+1,r)

- 对[l,r]分治的优势：**可能改变**的边权数是O(r-l)，可能远小于O(m)
  - 区间里的MST查询结果有相似性。

# [HNOI2010] City：Step 1

- 假设G是一个带权无向图的边集，S是G的一个子集。
- 将S中边权设为+inf后，T是图G的最小生成树。
  - 也可以说T是G-S的MST

- 定理1：不管S中的边权怎么变化，G的最小生成树T'将**属于**T∪S。
  - 对于任意一条不属于T∪S的边，我们可以在T中找到链接它两端的一条路径。
  - 由于这些边取值都和S无关，无论S权值怎么更改，这个环上这条边最大，不会进入MST

# [HNOI2010] City：Step 1

- 假设G是一个带权无向图的边集，S是G的一个子集。
- 将S中边权设为+inf后，T是图G的最小生成树。
  - 也可以说T是G-S的MST

- 定理2：在定理1的前提下，我们可以在不影响T'的情况下，将G的**边数**缩减到n+|s|-1以下。
  - 直接运用定理1，G-S的最小生成树最多有N-1条边。
  - 其他的边不可能在T'中，我们可以安全地删除掉。

- 这一步被称为Reduction，效果是减少了G的边数。
- 复杂度同MST是O(mlogm)，m=|G|。

# [HNOI2010] City：Step 2

- 假设G是一个带权无向图的边集，S是G的一个子集。
- 将S中边权设为-inf后，T是图G的最小生成树。

- 定理3：不管S中的边权怎么变化。G的最小生成树T’将**包含**T-S。
  - 考虑将S的权值一条边一条边提升的情况。
  - 每提升一条边权值，MST要么不变，要么就是S中的一条边离开，一条新边加入。
  - 无论如何，T-S这些边都不会离开MST。

# [HNOI2010] City：Step 2

- 假设G是一个带权无向图的边集，S是G的一个子集。
- 将S中边权设为-inf后，T是图G的最小生成树。

- 定理4：在定理3的前提下，我们可以在不影响T’的情况下，将G的**点数**缩减到|s|+1**以下**。
  - 假设已经对图进行了Reduction。
  - 根据定理3，由于T-S的边不离开MST，我们可以将这些边连接的点合并为联通分量，将联通分量视为节点。
  - 之后根据节点归属更新边表即可。
- 这一步被称为Contraction，效果是减少了G的点数。
- 复杂度同MST，O(mlogm)

# [HNOI2010] City

- 根据上面的推论，我们可以得到结论：
- 给定一个图G和操作序列S，可以在O(mlogm)时间内通过reduction-contraction将图的边数缩小到n+|s|-1，点数缩小到|s|+1而保持求解过程的正确性。
- 定义分治过程Solve(l,r)用于解决L到R区间的问题。
  - 为了方便起见，我们将图G作为参数传递。
- 定义分治过程Solve(l,r,G)解决L到R区间的问题。

# [HNOI2010] City

- Solve(l,r,G)
  - Reduction(G)
  - Contraction(G)
  - Solve(l,mid,G)
  - Solve(mid+1,r,G)
  - Recover(G) //用于撤销Reduction-Contraction的影响
- 边界情况：Solve(l,l,G)
  - 2个点和1条边！
  - 直接判断即可

# [HNOI2010] City

- 时间复杂度分析

- 在执行Solve(l,r,G)时，G的点数和边数都是O(r-l)的
  - 在第一次分治之前，对原图做一次R-C
  - 分治之后由归纳假设和R-C过程易得原证明成立

- R-C(n)的时间复杂度是O(nlogn)
- Recover的复杂度不会高于R-C(n)的复杂度
- $T(n)=2T(n/2)+O(n\log n)$

# Conclusion

- 基于时间（或阶段、顺序）分治
- 牺牲时间复杂度，将在线问题离线化
- 分治的递归过程用于解决段内问题，两段间问题由分治主过程解决
- 在特殊场合有很好的效果

- 优势：代码简短易读，效率不错
- 劣势：递归过程导致常数变大，有时候需要去递归

- （能够代替树套树这种数据结构的还有可持久化数据结构，有兴趣的同学可以去研究下。）

# [FJOI2012] Point

- 原题目较长，剔除边界情况后抽象成下列问题：
- 有长度为N的数组，开始时是空的。
- 执行N个操作Pi Qi，表示在第Pi位填上Qi。
- Pi,Qi是1到n的排列。
- 每个操作之后，输出序列的逆序对个数。n<=50000

- 用树套树、块状链表或者今天讲的分治方法，很容易做到O(nlog^2n)或相近的复杂度。考场上前两者都只拿到了50分。
- 有没有低于O(nlog^2n)的方法？比如O(nlogn)？